Improvement of the Tolerated Raw Bit Error Rate in NAND Flash-based SSDs with Selective Refresh

Emna Farjallah, Jean-Marc Armani, Valentin Gherman, Luigi Dilillo

Title: Improvement of the Tolerated Raw Bit Error Rate in NAND Flash-based SSDs with Selective Refresh

Authors: Emna Farjallah, Jean-Marc Armani, Valentin Gherman, and Luigi Dilillo

DOI: 10.1016/j.microrel.2019.01.014

Published: 3 April 2019

Document version: Post-print version (Final draft)

Please cite the original version: E. Farjallah, J.M. Armani, V. Gherman, L. Dilillo, “Improvement of the Tolerated Raw Bit Error Rate in NAND Flash-based SSDs with Selective Refresh”, Journal of Microelectronics and Reliability, Elsevier, 96, pp.37-45. DOI: 10.1016/j.microrel.2019.01.014, 2019.

Improvement of the Tolerated Raw Bit Error Rate in NAND Flash-based SSDs with Selective Refresh

Emna Farjallah, Jean-Marc Armani,
Valentin Gherman
CEA, LIST
Laboratoire Fiabilité et Intégration Capteurs
PC 172, 91191 Gif sur Yvette, France

Luigi Dilillo
LIRMM
UMR 5506, 161 rue Ada
34095 Montpellier Cedex 5, France

Abstract—**A recent large-scale study revealed that the uncorrectable bit error rates in data center solid-sate drives (SSDs) may fall far below the JEDEC standard recommendations. Here, a technique is proposed to improve the tolerated raw bit error rate (RBER) based on the observation that (a) a small SSD ratio may have a much higher RBER than the rest and (b) the RBER is dominated by the retention error rate. Instead of employing stronger but costly error-correcting codes an approach is used to estimate the remaining retention time, i.e., the reliable data storage time, of flash memory pages. This estimation can be performed each time a memory page is read based on the number of detected retention errors and the elapsed time since data was programmed. The fact that the estimated remaining retention time is smaller than a maximum time interval before the next read and check operation is an indication that data needs to be refreshed. It is estimated that the tolerated RBER can be increased by up to 35× over a storage period of 3 years if the stored data are checked on a monthly basis and refreshed only if necessary. The proposed technique has the ability to adapt the average time between refresh operations to the actual RBER. Maximum refresh time reductions of about 12x are reported as compared to systematic refresh schemes.**

Keywords—NAND flash; SSD; reliability; adaptability; data retention; bit error rate

I. Introduction

Solid-state drives (SSDs) based on NAND flash memories provide an attractive storage solution as they are faster and less power hungry than traditional hard-disc drives [13]. Unfortunately, the continuous technology scaling and emergence of flash memories with multilevel cells (MLC) brought not only cost per gigabit reductions but also reliability degradations. For instance, the cumulative number of program/erase (P/E) cycles that can be sustained by a flash memory cell, i.e., the *cycling endurance*, is decreased by an order of magnitude each time the cell storage capacity is enhanced with an additional bit [5][13][19]. What is more, a recent large-scale study revealed that the uncorrectable bit error rate (UBER) of data center SSDs can significantly exceed the JEDEC standard recommendations [12]. The reported UBER values are between 10^{-11} and 10^{-9} while client and enterprise class SSDs are required to provide an UBER below 10^{-15} and 10^{-16}, respectively [9].

An efficient approach to improve UBER is to use stronger error-correcting codes (ECCs). Unfortunately, powerful ECCs come with important storage and latency overheads. For instance, the storage overhead of a BCH code increases almost linearly with the number of correctable errors [7].

The need for strong ECCs may be reduced by containing the raw bit error rate (RBER). Besides technological fixes or solutions based on improved read and write algorithms [1][4][13], the RBER can be tempered if the stored data are periodically refreshed [2][13][14][17]. A refresh operation can be executed in-place by injecting only the missing amount of charge into the floating gate of the flash memory cells or by relocating the data to a different physical location [2][3]. Relocation operations may result in significant P/E cycle overhead especially in the case of read-intensive applications for which the data relocation frequency may become larger than the functional update rate [2]. A way to reduce this overhead is to adapt the relocation rate to the number of P/E cycles endured by each flash memory block [2][3].

Existent refresh schemes are based on worst-case scenarios, oblivious to intra- and inter-device variations, which may lead to unnecessary overheads with respect to response latency, dissipated power and P/E cycles. For example, the large-scale study reported in [12] unveiled that only a small number of SSDs may contribute to the overall UBER degradation.

This work proposes an approach to avoid the utilization of strong ECCs or worst-case refresh frequencies for dealing with a whole population of NAND flash memories or SSDs that may contain some error-prone units. The idea is to exploit the fact that the retention error rate dominates the RBER in NAND flash memories [3] and take advantage of flash read operations to estimate the remaining reliable retention time for each memory page. Such an estimation can be done based on the detected number of retention errors and the *retention age*, i.e., the elapsed time since data was programmed. A valid memory page needs to be refreshed only when its estimated remaining retention time is smaller than the time left to the next read and check operation.

Such a technique is effective when a maximum time interval is imposed between consecutive read and check operations of any memory page. For example, the tolerated RBER can be increased by up to 35× over a storage period of 3 years if one makes sure that the stored data are checked at least once in a month. The resulting data refresh frequency is not necessarily correlated with the data check frequency since it depends on the actual RBER. It is shown that the refresh probability is negligible at RBERs that can be managed by the available ECC alone and starts to increase only at larger RBERs. Compared to systematic refresh schemes able to ensure the same protection level, flash refresh time reductions of about 12× have been simulated.

The solution proposed here is orthogonal to other techniques used to reduce the number of retention errors based on the utilization of read reference voltages which are aware of (a) the data retention age [4] or (b) the number of bits vulnerable to retention errors [11].

Types of storage errors that may affect NAND flash memories are analyzed in Section II. The proposed refresh scheme based on the estimation of the remaining retention time of flash memory pages is presented in Section III. A method to reduce the number of check operations is proposed in Section IV. Simulation results concerning the improvement of the tolerated RBER and the reduction of the refresh frequency are reported in Section V. Concluding remarks are drawn in Section VI.

II. Typical Storage Errors in NAND Flash Memories

A flash memory cell consists of a MOS transistor with a supplementary floating gate or a charge trap layer embedded in the dielectric between channel and control gate. Data are programmed via the injection/erasure of electric charge into/from the floating gate or the charge trap layer. The threshold voltage distribution created by the injected charge into the floating gate of an MLC flash memory is illustrated in Fig. 1 [13]. In a NAND flash memory, between 32 and 64 memory cells are connected together to form a string. Thousands of strings are assembled in a storage array called block and few thousands of blocks may be contained in a flash memory chip. In a block, memory cells on the same string are accessed with the help of different word lines. The bits stored in memory cells accessed by the same word line are logically grouped into one or several pages.

A NAND flash memory can be affected by different types of storage errors like retention errors, write errors, also called program-interference or over-programming errors, read-disturb errors and erase errors. Retention errors affect the ability of a memory to keep the stored information over a required period of time. As shown in Fig. 2, retention errors appear due to a drift to the left of the threshold voltage distribution and the resulting crossing of the reference values used during read operations. For retention ages larger than one month, the retention error rate largely dominates the other error rates [3].

The remaining error types are characterised by a drift to the right of the threshold voltage distribution. Write errors are induced by parasitic capacitance-coupling affecting memory cells on a certain word line subsequent to a program operation on a neighbour word line. Once a memory block is fully programmed, the number of write errors does not increase with the retention age. In NAND flash memories, the write errors have the second largest occurrence rate [3].

Erase errors are the consequence of an erase operation that fails to reset all cells in a memory block to the erased state [3]. Upon the occurrence of an erase error an entire block may be marked as bad and discarded [13].

A read-disturb error occurs when the content of a memory cell is corrupted due to repeated read operations of cells on the same string. In the following, read-disturb errors will be neglected due to their very small occurrence rate [3].

Retention errors will be considered as the only storage errors whose rate may increase with the retention age once a memory bloc has been programmed. It will be assumed that the retention RBER ($RBER_{RET}$), i.e., the probability that a vulnerable bit is affected by a retention error, is given by the following expression:

$$RBER_{RET}(t_{AGE}) = 1 - e^{-\lambda t_{AGE}} \qquad (1)$$

where t_{AGE} is the retention age. The parameter λ is called failure rate and may vary from one SSD or flash memory to another [12][15], between the pages of the same memory block [3] and with the number of P/E cycles endured by a memory block [2][3]. This law is in agreement with reported results [15] and our own experience. The law also has the properties of a cumulative distribution function, i.e., $RBER_{RET}(0) = 0$ and $RBER_{RET}(\infty) = 1$.

Retention errors can be easily distinguished from other error types. Typically, each read operation is followed by an error correction step during which the erroneous bits are identified with the help of an ECC. The error-correcting process allows to infer the polarity of each error, i.e., the difference between the corrected and the initial values of the erroneous bit. The error polarity allows to identify the error type if one assumes that errors can only result from threshold voltage transitions between neighbouring states.

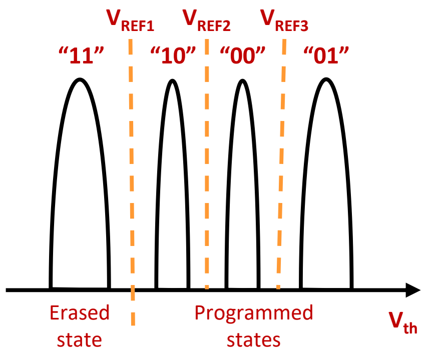
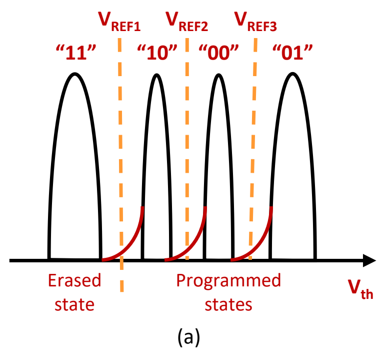
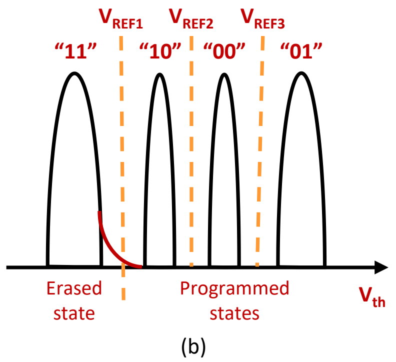


Fig. 1 Threshold voltage distribution and example of logical state encoding for a 2-bit MLC flash memory.





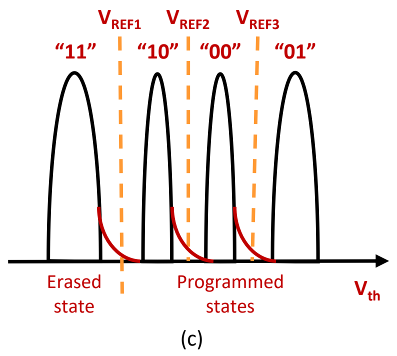


Fig. 2 Threshold voltage distribution of the logical states in a 2-bit MLC flash memory affected by (a) retention errors, (b) erase errors and (c) write and read-disturb errors.

Table I gives the retention error fingerprints for the MLC considered in Fig. 2. For flash memories with 1 bit per cell (SLC), the retention error fingerprint is given by the first line in Table I.

TABLE I. RETENTION ERROR FINGERPRINTS FOR THE MLC FLASH MEMORY CELL CONSIDERED IN FIG. 2.

Bits in the same cell	Read value	Corrected value	Value of the other bit
First bit	1	0	-
Second bit	1	0	1
Second bit	0	1	0

III. SELECTIVE REFRESH BASED ON A LINEAR APPROXIMATION OF THE RETENTION RBER

Here, an approach is proposed to deal with retention RBER variations in NAND flash memories beyond the error protection provided by an ECC. The main idea is to take advantage of read operations of any valid flash memory page to estimate its *left retention time* t_{LEFT} i.e. the storage time before the UBER target is exceeded [9]. Assuming a maximum time period T_{CHECK} between two consecutive read operations of any page, the read data has to be relocated or refreshed if t_{LEFT} is smaller than T_{CHECK}.

The proposed t_{LEFT} estimation is based on the observation that (1) can be approximated by the linear relation (2) since, in concrete situations, the product λt_{AGE} is much smaller than 1. For instance, if one considers an RBER equal to 10^{-3}, which is already a very large value, λt_{AGE} is equal to 10^{-3} with a difference of about 5×10^{-7} between (1) and (2). Smaller and more realistic RBER values correspond to smaller λt_{AGE} values and the relative error induced by (2) becomes much smaller.

$$RBER_{RET}(t_{AGE}) \cong \lambda t_{AGE} \tag{2}$$

Taking into account that RBER represents the number of retention errors divided by the number of vulnerable bits and neglecting the statistical variations that may affect the number of retention errors, one obtains:

$$\frac{M-n_{\neg RET}}{n_{RET}} = \frac{t_{AGE}+t_{LEFT}}{t_{AGE}} \tag{3}$$

where:

- n_{RET} represents the number of retention errors at the retention age t_{AGE}.
- $n_{\neg RET}$ stands for the number of non-retention errors that may appear during the programming of the block where the considered memory page is stored,
- M is an a priori known parameter and refers to the maximum number of errors that can be corrected with the available ECC,
- $M - n_{\neg RET}$ stands for the maximum number of retention errors that can be corrected in the considered memory page.

Based on (3), t_{LEFT} can be estimated as follows:

$$t_{LEFT} = \alpha_{DAMP} t_{AGE} \left(\frac{M-n_{\neg RET}}{n_{RET}} - 1\right) \tag{4}$$

where the damp factor α_{DAMP} is used to take into account statistical variations that may affect the number of retention errors. Here, α_{DAMP} values between 0.005 and 0.1 are used. In the absence of retention errors, i.e., $n_{RET} = 0$, t_{LEFT} is considered to be equal to the target retention time.

Neither the parameter λ nor the number of bits vulnerable to retention errors in the considered page does come out in the final expression and they do not need to be estimated in order to get t_{LEFT}.

The variables n_{RET} and $n_{\neg RET}$ can be calculated with the help of the ECC decoder. For example, the decoding scheme of a BCH ECC is usually concluded by the execution of a so-called *Chien algorithm* that checks each bit position for a potential error [7]. When an error location is found, a simple verification of the conditions in Table I allows to identify a retention or non-retention error. Two specially devoted counters can be used to keep track of n_{RET} and $n_{\neg RET}$.

The retention age t_{AGE} can be calculated as the difference between a timestamp associated to the page being accessed and the current state of the timer used to provide timestamps [6]. A single timestamp may be used to characterize the programing time of all pages in a flash memory block [18]. The resulting storage overhead of the timestamp table is significantly smaller than in the case of other metadata structures such as the remapping table of the flash translation layer [18].

The necessary check operations that should follow a page read operation are formalized in Algorithm 1 described below. A maximum time interval T_{CHECK} between consecutive check operations can be imposed via periodic scrubbing [16]. As it will be shown later, the tolerated RBER can be significantly improved when T_{CHECK} is equal to one or a few months.

Algorithm 1: Optimization of flash page refresh based on linear approximation of $RBER_{RET}$

```
Require: Read data from a flash memory page that has been decoded and corrected with the help of an ECC with known error correction strength
Require: $n_{RET}$, the number of already existing retention errors
Require: $n_{\neg RET}$, the number of already existing non-retention errors
Require: $T_{CHECK}$, the maximum time interval between consecutive check operations
1  Calculate the retention age $t_{AGE}$ of the accessed page
2  Get the remaining retention time $t_{LEFT}$ as a function of $t_{AGE}$, $n_{RET}$ and $n_{\neg RET}$ such that the UBER target is still preserved
3  if $t_{LEFT} < T_{CHECK}$ then
4      Refresh the considered page
5  end
```

The remaining retention time t_{LEFT} can be computed on-line or off-line for all possible n_{RET} and $n_{\neg RET}$ combinations. In the latter case, the storage overhead can be reduced by observing that t_{LEFT} decreases with n_{RET} and one only needs to store the largest n_{RET} value with t_{LEFT} larger than T_{CHECK}. In such a case, the line 3 in Algorithm 1 may be implemented as the comparison between the maximum tolerated n_{RET} and the measured n_{RET}. The cost of storing the maximum tolerated n_{RET} does not depend on the storage capacity of the protected SSD or flash memory system. This cost increases proportionally with $1/T_{CHECK}$ and with the maximum number of errors that can be corrected by the available ECC. For example, this cost amounts to a few hundred bits with an ECC able to correct up to 10 errors per page and T_{CHECK} equal to 1 month.

Refresh operations have a negative impact on the average response time of a flash memory system or SSD. It has been reported that this impact can be reduced by increasing the refresh interruption granularity [17]. The performance overhead of the proposed refresh scheme is expected to be lower since not all check operations have to be followed by a refresh operation. Furthermore, a method to decrease the number of check operations is proposed in the following section.

IV. Reduction of the Number of Check Operations

Check operations as described in Algorithm 1 can be imposed periodically via warnings triggered by a conventional or temperature aware timer [18]. Since such checks can also be associated to functional read operations, frequently accessed pages may be skipped during the periodical checks triggered by the timer.

This can be achieved with the help of one flag bit per flash memory page that indicates whether the page has already been checked. The page flag is set to 0 each time a page is read, programmed or refreshed as illustrated in Fig. 3. After the first subsequent warning, the page does not need to be checked, only its flag is set to 1. After a second warning, the page has to be checked and its flag set to 0 only if the page is still valid and if no functional read operation occurred between the first and second warnings. Otherwise, only the page flag will be set to 1. Invalid pages can be neglected and the warnings need to be triggered with a period $T_{WARNING}$ which is equal to $T_{CHECK}/2$.

In order to improve performance with the proposed method, the page flags have to be kept in the RAM memory of the SSD controller and not in the relatively slower flash memory. This method has no storage overhead since the page flags do not need to be stored but only initialized to 1 at system start-up.

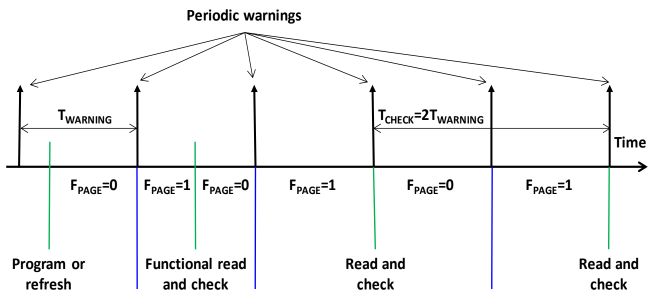


Fig. 3 Management of page flags F_{PAGE} used to reduce the number of page read and checks as decribed in Algorithm 1.

V. Simulation Results

In order to assess the effectiveness of the proposed approach, we first evaluated the increase of the tolerated $RBER_{RET}$ with respect to the case when the stored data is not refreshed. We considered data characterized by a small functional update frequency and, implicitly, very large retention age t_{AGE}. Such data may contain user files, executable files and operating system files.

The obtained results are reported in Table II for different numbers of correctable errors per page. The tolerated $RBER_{RET}$ values are compliant with the requirement to keep UBER below 10^{-16} [9]. UBER calculation details are given in Annex I and Annex II. It should be noted that UBER is a bit-level observable that only depends on $RBER_{RET}$, page size, ECC strength, refresh frequency and numbers of non-retention errors and bits vulnerable to retention errors in a page. The impact of the page size is not investigated since it affects UBER only as a scaling factor. Only 2KB pages have been considered in the most unfavourable case when all bits are vulnerable to retention errors.

As illustrated in Table II, the tolerated $RBER_{RET}$ is increased by the reduction of T_{CHECK}, i.e., the maximum time interval between two successive check operations of flash memory pages. When T_{CHECK} is equal to 1 month, the obtained improvement factors are between 32× and 35×. Better results should be expected for lower T_{CHECK} values. Moreover, the maximum tolerated $RBER_{RET}$ values and resulting improvement factors are pessimistic as check operations triggered by functional read operations are not taken into account.

A T_{CHECK} equal to 1 month may have a larger impact on the tolerated $RBER_{RET}$ than a multiplication by four of the number of correctable errors. This can be observed by comparing the *1 month* column for 10 correctable errors with the *no check* column for 40 correctable errors. A T_{CHECK} value equal to 6 months may provide a larger improvement of the tolerated $RBER_{RET}$ than a duplication of the number of correctable errors. This can be verified by comparing the *6 months* column for 20 correctable errors with the *no check* column for 40 correctable errors. For BCH codes, the increase of the number of correctable errors from 20 to 40 may augment the storage overhead by 75%. This overhead may become 230% when the number of correctable errors is changed from 10 to 40.

The results reported here can only be expected for systems with downtimes much shorter than the imposed check period T_{CHECK}. This is true for enterprise class SSDs with limited outage periods, i.e., maximum few hours per year [19]. When longer outage periods are expected, one has to consider the maximum downtime $T_{POWER-OFF}$ and modify the *if condition* in Algorithm 1 such that a data refresh operation is initiated if $t_{LEFT} < T_{CHECK} + T_{POWER-OFF}$. In such a case, the benefit brought by the proposed approach can be calculated by considering an imposed check period equal to $T_{CHECK} + T_{POWER-OFF}$. For instance, if both T_{CHECK} and $T_{POWER-OFF}$ are 1 month, the improvements obtained with our approach are given by the *2 months* column in Table II. The case when $T_{POWER-OFF}$ is equal to 3 months corresponds to a JEDEC requirement for enterprise class SSDs [9]. In Table II, the *4 months* column gives worst-case improvement factors larger than 8× for JEDEC compliant enterprise class SSDs in which data are checked every month.

The reported improvements of the tolerated $RBER_{RET}$ require a number of refresh operations that may be greatly reduced as compared to a conventional scheme with systematic refresh operations [2][3]. As illustrated in Fig. 4, when the check period is 1 month, the average time between refresh operations may be significantly improved as compared to the case when the data are systematically refreshed, not to speak of the difficulty to infer an ideal refresh period at run-time for a scheme with systematic refresh operations.

A larger average time between refresh operations enables a reduction of the time during which the flash chips are accessed for refresh operations. Compared to a systematic refresh scheme with fixed refresh frequency, the reduction of the time spent for refresh-triggered read and write operations can be expressed as follows:

$$\frac{T_{SYS\_REFRESH}}{T_{REFRESH}} = \frac{(\tau_{WR}+\tau_{RD})\, D_{SYS\_REFRESH}\, f_{SYS\_REFRESH}}{\tau_{RD}\, D_{CHECK}\, f_{CHECK}+\tau_{WR}\, D_{REFRESH}\, f_{REFRESH}}$$

where:

- $T_{REFRESH}$ and $T_{SYS\_REFRESH}$ represent the overall times spent for refresh-triggered read and write operations with the proposed and systematic refresh schemes,

TABLE II. IMPROVEMENT OF THE MAXIMUM RETENTION RBER THAT CAN BE TOLERATED IN A 2KB FLASH MEMORY PAGE. THE REPORTED RBER FIGURES CORRESPOND TO A TARGET RETENTION TIME OF 36 MONTHS. IT IS CONSIDERED THAT ALL BITS IN A PAGE ARE VULNERABLE TO RETENTION ERRORS AND THAT THERE ARE 0 NON-RETENTION ERRORS.

Number of correctable errors per page	Maximum tolerated $RBER_{RET}$ with UBER ≤ 10^{-16}						Improvement factor with respect to no refresh				
		check period (T_{CHECK})					check period (T_{CHECK})				
	no check	6 months	4 months	3 months	2 months	1 month	6 months	4 months	3 months	2 months	1 month
10	2.64×10^{-5}	1.44×10^{-4}	2.14×10^{-4}	2.85×10^{-4}	4.26×10^{-4}	8.52×10^{-4}	5.5	8.1	10.8	16.1	32.3
20	1.65×10^{-4}	9.62×10^{-4}	1.42×10^{-3}	1.89×10^{-3}	2.83×10^{-3}	5.65×10^{-3}	5.8	8.6	11.5	17.2	34.2
30	3.84×10^{-4}	2.21×10^{-3}	3.32×10^{-3}	4.42×10^{-3}	6.63×10^{-3}	1.32×10^{-2}	5.8	8.6	11.5	17.3	34.4
40	6.56×10^{-4}	3.89×10^{-3}	5.82×10^{-3}	7.76×10^{-3}	1.16×10^{-2}	2.31×10^{-2}	5.9	8.9	11.8	17.7	35.2

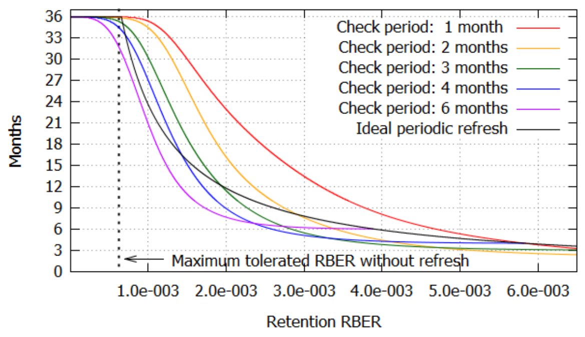


Fig. 4 Average time between refresh operations with the proposed scheme and an ideal systematic refresh scheme with the refresh frequency adapted to the actual RBER. We considered flash memory pages with 2KB bits and up to 40 correctable errors. Each colored curve stops at the maximum retention RBER that can be tolerated.

- τ_{WR} and τ_{RD} stand for the latencies of page write and read operations,
- $f_{SYS\_REFRESH}$ and $D_{SYS\_REFRESH}$ represent the fixed refresh frequency and the amount of data that needs to be systematically refreshed with a conventional refresh scheme,
- f_{CHECK} and $f_{REFRESH}$ are the check frequency and the average refresh frequency of the proposed refresh scheme,
- D_{CHECK} and $D_{REFRESH}$ represent the amounts of data that are checked and refreshed with the proposed scheme.

With the method proposed in Section IV to reduce the amount of checked data, D_{CHECK} will include all data with a retention age larger than $T_{WARNING}$ in Fig. 3. The same is true for $D_{SYS\_REFRESH}$ when the method illustrated in Fig. 3 is adapted to implement a systematic refresh scheme [8]. In our experiments, α_{DAMP} in (4) is selected such that the maximum tolerable $UBER$ is reached at an $RBER_{RET}$ for which the average retention time is equal to $T_{CHECK} = 1/f_{CHECK}$. Since the average retention time can be expressed as $1/f_{SYS\_REFRESH}$, $f_{SYS\_REFRESH}$ and f_{CHECK} are equal. As a consequence, $D_{SYS\_REFRESH}$ and D_{CHECK} will be equal as well. Since $D_{REFRESH}$ is smaller or equal to D_{CHECK}, a lower bound for the reduction of the time spent for refresh-triggered read and write operations can be defined as follows:

$$\frac{T_{SYS\_REFRESH}}{T_{REFRESH}} \geq \frac{(\tau_{WR}+\tau_{RD})\, f_{SYS\_REFRESH}}{\tau_{WR} f_{REFRESH}+\tau_{RD} f_{CHECK}} \qquad (5)$$

Figures 5 to 8 illustrate estimations of the lower bound used in (5) for MLC NAND flash chips with average write and read latencies equal to $975\mu s$ and $50\mu s$ [10], respectively. Maximum reductions of about 12× are obtained for retention RBER values that actually do not require refresh operations. For flash memory and SSD populations where only a small number of units are error-prone [12], the average reduction of the time spent for refresh-triggered read and write operations is expected to approach the maximum reduction values near the vertical dashed lines in figures 5 to 8. Even better results

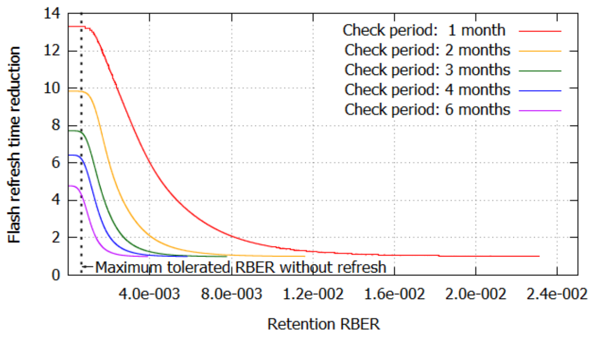


Fig. 5 Reduction of the overall time spent for refresh-triggered read and write operations compared to a systematic scheme with fixed refresh frequency. Each curve stops at the maximum tolerated retention RBER. The considered parameters are the same as those used in Fig. 4.

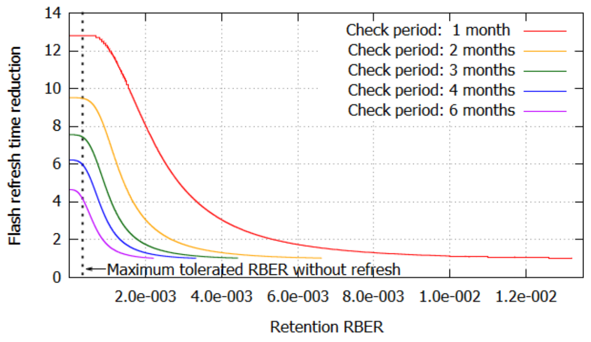


Fig. 6 Similar to Fig. 5 but for an ECC able to correct up to 30 errors per flash memory page.

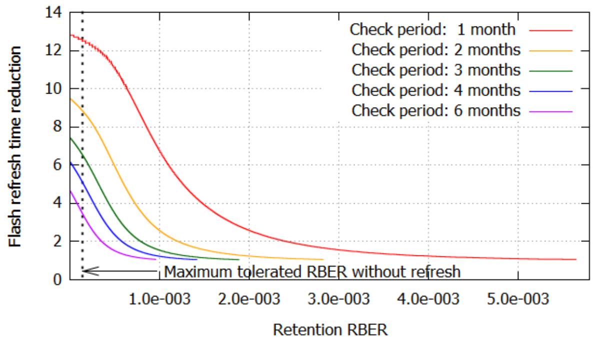


Fig. 7 Similar to Fig. 5 but for an ECC able to correct up to 20 errors per flash memory page.

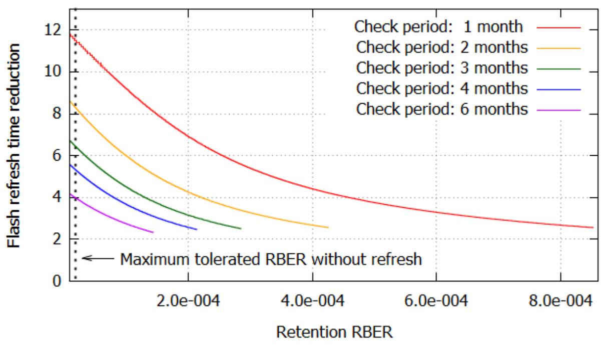


Fig. 8 Similar to Fig. 5 but for an ECC able to correct up to 10 errors per flash memory page.

can be obtained for larger τ_{WR}/τ_{RD} ratios.

When up to 10 errors per page can be corrected, the reductions of the time spent for refresh operations are smaller and the curve shapes are different as illustrated in Fig. 8. This is due to the fact that in this case refresh operations are avoided only in the absence of retention errors. This is also the reason for the slightly smaller improvement factors reported in Table II.

Here, the impact of ECC encoding and decoding on refresh latency is not explicitly considered as these steps can be performed in parallel to flash chip operations. For BCH codes and other linear block ECCs the encoding latency is relatively small. Moreover, during triggered check operations the ECC decoding process does not need to be completed if the number of detected errors does not require a refresh operation. For example in the case of a BCH, the number of errors is calculated before the relatively long error correction step [7]. This means that the ECC decoding latency is relatively small when the number of errors and the refresh probability are low. Particularly, the absence of errors can be rapidly detected by checking whether the generated syndrome is an all-zero vector.

The discussed refresh schemes may also trigger erase operations when data refresh is based on data relocations. One can estimate that the number of erase operations is proportional to the amount of data that needs to be refreshed and, implicitly, relocated. In this case, the reduction of the number of refresh-triggered erase operations can be expressed as shown below:

$$\frac{E_{SYS\_REFRESH}}{E_{REFRESH}} = \frac{D_{SYS\_REFRESH} f_{SYS\_REFRESH}}{D_{REFRESH} f_{REFRESH}}$$

where $E_{REFRESH}$ and $E_{SYS\_REFRESH}$ represent the numbers of refresh-triggered erase operations with the proposed and systematic refresh schemes.

As long as $f_{REFRESH}$ is smaller than $f_{SYS\_REFRESH}$, $D_{REFRESH}$ is also smaller or equal to $D_{SYS\_REFRESH}$ and a lower bound for the reduction of the number of refresh-triggered erase operations can be defined as follows:

$$\frac{E_{SYS\_REFRESH}}{E_{REFRESH}} \geq \frac{f_{SYS\_REFRESH}}{f_{REFRESH}} \quad (6)$$

Estimations of the lower bound used in (6) are illustrated in figures 9 to 12. As with the read and write operations, maximum reductions of the number of refresh-triggered erase operations, i.e. larger than 30×, are obtained for retention RBER values that actually do not require refresh operations. At such RBER values, one could expect that the reduction would become very high or reach infinity. This is not the case as retention errors may still occur and trigger refresh operations even with the proposed method. The reductions could be improved by increasing the value of α_{DAMP} in (4) at the cost of a smaller tolerated retention RBER.

The number of refresh-triggered erase operations and, implicitly, the time required for the execution of such operations

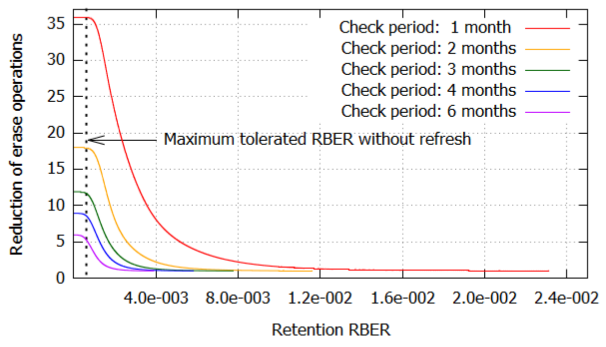


Fig. 9 Reduction of the number of refresh-triggered erase operations compared to a systematic scheme with fixed refresh frequency. Each curve stops at the maximum tolerated RBER. The considered parameters are the same as those used in Fig. 5.

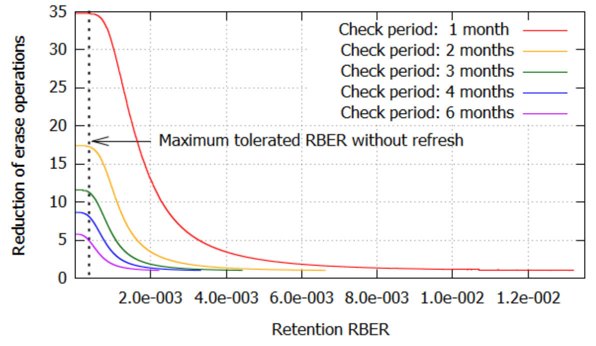


Fig. 10 Similar to Fig. 9 but for an ECC able to correct up to 30 errors per flash memory page.

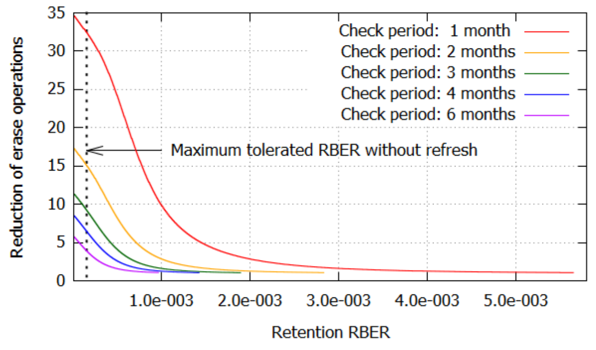


Fig. 11 Similar to Fig. 9 but for an ECC able to correct up to 20 errors per flash memory page.

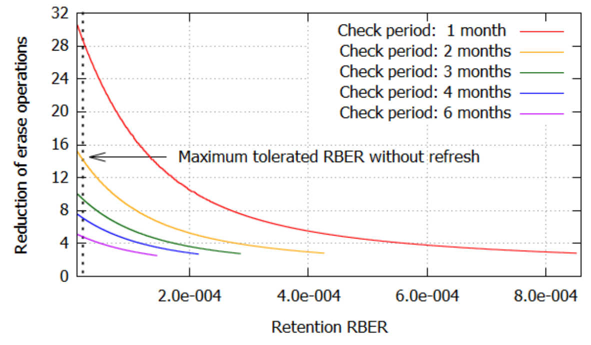


Fig. 12 Similar to Fig. 9 but for an ECC able to correct up to 10 errors per flash memory page.

is reduced to a larger extent than the time spent for refresh-triggered read and write operations. This means that the figures reported for the reduction of the read and write operations can be used as a lower bound for the reduction of the time spent for all three types of refresh-triggered operations.

VI. Conclusions

An approach was proposed to improve the tolerated raw bit error rate (RBER) in NAND flash-based SSDs via an estimation of the remaining retention time. This estimation can be performed each time a flash memory page is read and relies on the number of detected retention errors and the calculated retention age, i.e., the elapsed time since data was programmed. The checked data should be refreshed if the estimated remaining retention time is smaller than a maximum time period to the next read operation. Improvement factors of the tolerated retention error rate between 32× and 35× have been simulated for data checked on a monthly basis over a storage period of 3 years. Such an improvement may be larger than what can be obtained by using an ECC able to correct 4× more errors. Even for a check period of 6 months the tolerated RBER improvement can be larger than when the ECC strength is doubled. The proposed method has the ability to adapt the average time between refresh operations to the actual retention RBER. This enabled maximum refresh time reductions of about 12× as compared to systematic refresh schemes.

Annex I

In the absence of refresh operations, the uncorrectable bit error rate (UBER) of a flash memory page can be computed with the expression below based on the assumption that only retention errors may accumulate in time [15].

$$UBER(t_{AGE}) = \frac{1}{N} \sum_{n_{RET}=M-n_{\neg RET}+1}^{N_{VUL}} \binom{N_{VUL}}{n_{RET}} \left(RBER_{RET}(t_{AGE})\right)^{n_{RET}} \times (1 - RBER_{RET}(t_{AGE}))^{N_{VUL}-n_{RET}}$$

where:

- t_{AGE} represents the storage period,
- N is the total number of bits in a flash memory page,
- M is the maximum number of errors that can be corrected with the available ECC,
- N_{VUL} is the actual number of bits vulnerable to retention errors in the considered flash memory page,
- n_{RET} and $n_{\neg RET}$ are the numbers of bits affected by retention and non-retention errors in the considered flash memory page,
- $M - n_{\neg RET}$ represents the maximum number of retention errors that can be corrected with the available ECC in the considered flash memory page,
- $RBER_{RET}(t_{AGE})$ is calculated according to (1).

Annex II

For flash memories with pages that are periodically checked and may be refreshed according to Algorithm 1, UBER can be obtained by adding the probabilities of the uncorrectable errors that may occur in a flash memory page between consecutive check operations as follows:

$$UBER = \frac{1}{N} \sum_{i=1}^{\left\lceil \frac{T_{MAX}}{T_{CHECK}} \right\rceil} UBER(i)$$

where:

- N is the total number of bits in a flash memory page,
- T_{MAX} is the maximum required retention time,
- T_{CHECK} is the time interval between consecutive triggered check operations,
- $UBER(i)$ represents the contribution to UBER of the uncorrectable errors that may occur during the time interval between the *(i-1)th* and *ith* triggered check operations,
- $\lceil\rceil$ stands for the ceiling function.

UBER(i) can be calculated with the relation below that takes into account the occurrence probabilities of all retention error numbers which do not impose a page refresh according to Algorithm 1 during the *(i-1)th* check operation and (b) the probability that during the *ith* triggered check operation the errors cannot be corrected anymore:

$$UBER(i) = \sum_{n_{RET}=0}^{n_{i-1}} P((i-1) * T_{CHECK}, N_{VUL}, n_{RET}) \times \left[1 - \sum_{n'_{RET}=0}^{M-n_{\neg RET}-n_{RET}} P(T_{CHECK}, N_{VUL} - n_{RET}, n'_{RET})\right] \quad (7)$$

where:

- $P((i-1) * T_{CHECK}, N_{VUL}, n_{RET})$ is the occurrence probability of n_{RET} retention errors that do not impose a page refresh according to Algorithm 1 during the *(i-1)th* check operation,
- n_{i-1} is the maximum number of retention errors for which the *if condition* in Algorithm 1 is false and no refresh operation needs to executed during the *(i-1)th* check operation,
- $P(T_{CHECK}, N_{VUL} - n_{RET}, n'_{RET})$ is the occurrence probability of n' retention errors that can still be handled by the available ECC during the *ith* check operation,
- M is the maximum number of errors that can be corrected by the available ECC,

- N_{VUL} is the actual number of bits vulnerable to retention errors in the considered flash memory page,
- $n_{\neg RET}$ is the number of non-retention errors in the considered flash memory page,
- n_{RET} and n'_{RET} are numbers of retention errors that can be tolerated in the considered flash memory page.

The probability to have n_{RET} retention errors in a flash memory page with N_{VUL} vulnerable bits after a storage period $i * T_{CHECK}$ can be calculated recursively as follows:

$$P(i * T_{CHECK}, N_{VUL}, n_{RET}) = \sum_{n'_{RET}=0}^{\min(n_{RET}, n_{i-1})} P((i-1) * T_{CHECK}, N_{VUL}, n'_{RET}) \times P(T_{CHECK}, N_{VUL} - n'_{RET}, n_{RET} - n'_{RET})$$

where:

- each term represents the probability of a possible repartition of n_{RET} retention errors over the time period before the *(i-1)*th check operation and the time interval between the *(i-1)*th and *i*th check operations,
- as in (7), n_{i-1} indicates that not all error occurrence scenarios are possible due to a refresh operation that may be triggered during the execution of the *(i-1)*th check operation,
- for $i = 1$, one can use the expression below:

$$P(T_{CHECK}, N_{VUL}, n_{RET}) = \binom{N_{VUL}}{n_{RET}} \left(RBER_{RET}(T_{CHECK})\right)^{n_{RET}} \times \left(1 - RBER_{RET}(T_{CHECK})\right)^{N_{VUL}-n_{RET}}$$

REFRENCES

[1] D. Bertozzi et al., “Performance and reliability analysis of cross-layer optimizations of NAND flash controllers,” ACM Transactions on Embedded Computing Systems, vol. 14, no. 1, Article 7, 2015.

[2] Y. Cai et al., “Flash correct-and-refresh: retention-aware error management for increased flash memory lifetime,” IEEE International Conference on Computer Design, pp. 94–101, 2012.

[3] Y. Cai et al., “Error analysis and retention-aware error management for nand flash memory,” Intel Technology Journal, Volume 17, Issue 1, pp. 140–164, 2013.

[4] Y. Cai et al., “Data retention in MLC NAND flash memory: characterization, optimization, and recovery,” International Symposium on High Performance Computer Architecture, pp. 551-563, 2015.

[5] L.-P. Chang, “Hybrid solid-state disks: combining heterogeneous NAND flash in large SSDs,” IEEE Asia and South Pacific Design Automation Conference, pp. 428–433, 2008.

[6] S. Di Carlo et al. “FLARES: an aging aware algorithm to autonomously adapt the error correction capability in NAND flash memories,” ACM Transactions on Architecture and Code Optimization, vol. 11, issue 3, article no. 26, Oct. 2014.

[7] E. Fujiwara, “Code design for dependable systems: theory and practical applications,” Wiley-Interscience, pp. 60, 2006, ISBN: 0471756180.

[8] V. Gherman, E. Farjallah, J.-M. Armani, M. Seif, and L. Dilillo, "Improvement of the Tolerated Raw Bit Error Rate in NAND Flash-based SSDs with the Help of Embedded Statistics," IEEE International Test Conference, pp. 1-9, 2017.

[9] JEDEC Standard, “Solid-state drive (SSD) requirements and endurance test method,” JESD218A, February 2011.

[10] X. Jimenez, D. Novo, and P. Ienne, “Wear Unleveling: Improving NAND Flash Lifetime by Balancing Page Endurance,” USENIX Conference on File and Storage Technologies, pp. 47–59, 2014.

[11] C. Lee et al., “A 32Gb MLC NAND-flash memory with V_{th}-endurance-enhancing schemes in 32nm CMOS,” ISSCC, pp. 446–448, 2010.

[12] J. Meza et al., “A large-scale study of flash memory failures in the field,” ACM SIGMETRICS International Conference on Measurement and Modeling of Computer System, pp. 177–190, 2015.

[13] R. Micheloni, L. Cripa, and A. Marelli, “Inside NAND flash memories,” Springer-Verlag, Berlin, 2010.

[14] Micron Technology, “TN-12-30: NOR flash cycling endurance and data retention introduction,” Technical Note, 2013.

[15] N. Mielke et al., “Bit error rate in NAND flash memories,” IEEE Annual International Reliability Physics Symposium, pp. 9–19, 2008.

[16] S. Mukherjee et al., "Cache scrubbing in microprocessors: myth or necessity?," IEEE Dependable Computing, 2004.

[17] Y. Pan et al., “Quasi-nonvolatile SSD: trading flash memory nonvolatility to improve storage system performance for enterprise applications,” High Performance Computer Architecture, pp. 1–10, 2012.

[18] M. Seif et al., "Refresh frequency reduction of data stored in SSDs based on A-timer and timestamps," IEEE European Test Symposium, pp. 1–6, 2017.

[19] N. Sundby and D. Taylor, “Beyond capacity: storage architecture choices for the modern datacenter,” White Paper, IDC Analyze the Future, Sponsored by: Toshiba, February 2014.